



## yooShell : yoo can do the command line.

---

Thank you for choosing yooShell from yooPlugs. This manual provides instructions for installing and using your FileMaker plug-in.

yooShell brings the power of the *Terminal* and *Command Prompt* to FileMaker.

Many people are surprised to learn of the power lurking beneath the surface of Mac OS X and Windows. Under their graphical interfaces, these operating systems are built to support textual commands that run programs of all sorts, inspect and configure the system, and find, create, and manipulate files and folders.

Generically, such commands are said to be executed *at the command line*. An environment in which such commands are executed is commonly called a *shell*.

yooShell unlocks the hidden power of your computer, giving you complete access to the command line. There is literally no limit to the new functionality that yooShell makes available in FileMaker.

- Perform custom calculations with command-line utilities
- Process & analyze your data with external tools
- Run custom scripts
- Find files on the computer

It can be a thrill to discover the things you never knew your computer could do at no additional cost. From simply reading a text file, to sending email, encrypting data, or running advanced statistical analysis, yooShell puts the countless possibilities at your disposal in FileMaker. Mac OS X users will especially benefit from the wealth of command-line Unix utilities built in to every system.

Developers using yooShell gain the freedom to write custom command-line utilities in the language of their choice. Using yooShell, even the simplest command-line software can be used to enhance FileMaker in ways as yet unimagined.

yooShell removes the barrier between FileMaker and the rest of your system. The number of tools you can bring to bear on your FileMaker databases just got a whole lot bigger.

# Installing yooShell

---

In order to use yooShell, you must first copy it into the *Extensions* folder inside the FileMaker folder. The location of the FileMaker folder may vary, but Macintosh users will typically find it in the *Applications* folder, and Windows users will typically find it in the *Program Files* folder.

If FileMaker is running when you install yooShell, you will need to quit FileMaker and run it again before you can use yooShell.

## Macintosh Users

Assuming you downloaded yooShell, you should have a file named yooShell-1.0.dmg. This disk image contains the file **yooShell.fmplugin**, which is the yooShell plug-in. Copy this file into FileMaker's *Extensions* folder.

*[ Note: yooShell-1.0.dmg is a "disk image" which may have automatically "mounted" on your desktop when you downloaded it. If not, double-click the disk image to mount the disk named yooShell-1.0 on your desktop. Mounting the disk in this way is equivalent to inserting a CD-ROM; when you are done installing yooShell, you "unmount" the disk the same way you would eject a CD-ROM. ]*

## Windows Users

Assuming you downloaded yooShell, you should have a file named yooShell-1.0.zip. This zip file contains a folder named *yooShell-1.0*. Inside the *yooShell-1.0* folder is the file **yooShell.fmx**, which is the yooShell plug-in. Copy this file into FileMaker's *Extensions* folder.

*[ Note: yooShell-1.0.zip may be automatically unzipped when you download it. If not, double-clicking the file should unzip the file, and create the yooShell-1.0 folder. If you cannot unzip the file, download a freely available utility, such as Stuffit Expander, that is capable of unzipping the file. ]*

# Calling Functions in yooShell

---

The functions in yooShell can be accessed anywhere the *Specify Calculation* dialog is available in FileMaker. The most common method of calling a yooShell function is as part of the *Set Field* script step. All of the examples in this document are presented in this context. Consult the FileMaker documentation for more information on creating scripts using ScriptMaker.

Inside the *Specify Calculation* dialog, selecting "External functions" from the *View* drop-down menu will display a list of functions grouped by plug-in name. Double-clicking on one of yooShell's functions will produce a template for your function call, including a placeholder for each parameter that the function accepts.

You should replace each parameter's placeholder with an appropriate value. The values you provide can be either database fields selected from the list in the *Specify Calculation* dialog, or literal values that you enter manually (for example, numbers like 23, or text like "hello").

## Optional Parameters

Any placeholders at the end of the list that are enclosed in square brackets correspond to optional parameters. You should delete the square brackets, as well as the placeholders for any of the optional parameters which you do not provide.

Note that optional parameters can only be omitted from the *end* of the parameter list. For example, suppose there are 3 optional parameters, and you are only interested in specifying the second one. You can omit the third optional parameter because it is at the end, but you cannot specify the second optional parameter without also specifying the first. Unless otherwise noted in the function reference, you can typically provide an empty parameter like "" in situations that force you to specify a parameter for which you do not have a meaningful value to provide.

If any optional parameters are omitted, the last parameter that you do provide must be followed with a semicolon (see examples in the function reference). Failure to use the semicolon will result in complaints from FileMaker such as "There are too few parameters in this function."

## Data Returned from Functions

As previously noted, the examples in this document assume that yooShell functions are being called as part of a *Set Field* script step. If this is the case, a complete script step will be of the general form:

```
Set Field [myDB::myField1; yooShell_function( myDB::myField2; )]
```

This will store the data returned from the call to `yooShell_function` in the database field `myDB::myField1`. However, there may be circumstances in which you are not interested in the returned data. Further, some functions in yooShell return no data at all. In such circumstances, it is common practice to define a "garbage" database field which is not part of the visual layout. Such a field can be used as the target of *Set Field* whenever it is desirable to effectively discard the data returned from a function call.

# Function Reference

---

The following reference observes various typographic and presentational conventions to help you quickly see important information.

- Each **basic** and **advanced** function has its own subsection beginning with a header consisting of a colored box containing the function entry as it appears in FileMaker's *External Functions* list.
- **Bold** letters are used for function names.
- [ Square brackets ] surrounding parameters at the end of the parameter list in a function header denote that the parameters are optional.
- **Bold blue** letters are used for a function's required parameters.
- **Bold gray** letters are used for a function's optional parameters.
- "Literal" text, which appears in red, should be passed to yooShell exactly as it appears in the reference.
- *<Qualitative>* text, which appears in italicized green, describes the nature of the value that should be passed to yooShell. For example, where the reference specifies *<ZIP Code>*, one might pass the value "52246" to yooShell.

## Important Reminder: Optional Parameters

Many functions in yooShell feature optional parameters. Frequently, some or all of these optional parameters can be omitted. When passing parameters to yooShell, remember two things:

1. If any optional parameters are omitted, you must follow the last parameter you provide with a semicolon.
2. You can only omit optional parameters from the *end* of the parameter list.

The functions in yooShell are here divided into two categories: the basic functions which offer essential functionality for every user, and the advanced functions which offer extra power to users with more complex requirements.

### BASIC

- yooShell\_exec** – Execute a command and retrieve its output.
- yooShell\_pwd** – Query the current working directory.
- yooShell\_cd** – Change the current working directory.
- yooShell\_getEnv** – Query the value of an environment variable.
- yooShell\_setEnv** – Set the value of an environment variable.

### ADVANCED

- yooShell\_config** – Inspect and adjust yooShell's configuration.

## yooShell\_exec( **command** )

### Parameters

**command** — The command to execute.

### Data Returned to FileMaker

The standard output of the given command.

### Examples (Macintosh)

```
Set Field [myDB::myText; yooShell_exec( "ls -l" )]
```

```
Set Field [myDB::myText; yooShell_exec( "find ~/Desktop" )]
```

```
Set Field [myDB::myText; yooShell_exec( "echo 'Sibilance.' | mail -s 'Testing 1-2-3' jdoe@anonymous.com" )]
```

```
Set Field [myDB::myText; yooShell_exec( "open http://www.yooplugs.com" )]
```

### Examples (Windows)

```
Set Field [myDB::myText; yooShell_exec( "dir" )]
```

```
Set Field [myDB::myText; yooShell_exec( "date /T" )]
```

```
Set Field [myDB::myText; yooShell_exec( "nslookup www.filemaker.com" )]
```

```
Set Field [myDB::myText; yooShell_exec( "start http://www.yooplugs.com" )]
```

### Description

The **exec** function is the heart of yooShell. Use it to execute the same commands that can be executed via the *Terminal* utility on Mac OS X, or the *Command Prompt* utility on Windows. The text that is produced as a result of executing the command is captured and returned to FileMaker.

Every command is executed in the context of the current *working directory* (see the **pwd** and **cd** functions), and the current *environment* (see the **getEnv** and **setEnv** functions). Changes to the working directory or the environment made by the given command affect only the current call to **exec**.

An important guideline is that the **exec** function should only be used to execute commands that can run to completion without outside intervention. If this guideline is not observed, it is possible to create a situation in which the command cannot complete without input that cannot be provided; hence FileMaker will be unable to continue, and it will need to be forcibly terminated.

## yooShell\_pwd

### Data Returned to FileMaker

The full path of the current working directory.

### Examples

Set Field [myDB::myText; yooShell\_pwd]

### Description

The **pwd** function identifies FileMaker's current working directory. All commands executed using the **exec** function will start in the directory returned by **pwd**. Which particular directory is identified by **pwd** can only be changed using the **cd** function.

As a simple example of when the working directory is significant, imagine using the **exec** function to get a listing of directory contents (using the *ls* command on Mac OS X, or the *dir* command on Windows). By default, these commands report the contents of the working directory.

More generally, the working directory is significant whenever relative paths are used. A full path completely specifies the location of a file or folder. Frequently it is useful to use partial paths to specify the location of a file or folder. Such paths are said to be relative because they identify the location of the file or folder relative to the working directory.

## yooShell\_cd( **directory** )

### Parameters

**directory** — The path to the new working directory.

### Data Returned to FileMaker

No data is returned to FileMaker.

### Examples

Set Field [myDB::myVoid; yooShell\_cd( "." )]

Set Field [myDB::myVoid; yooShell\_cd( "Some Subdirectory" )]

Set Field [myDB::myVoid; yooShell\_cd( "/Applications" )]

Set Field [myDB::myVoid; yooShell\_cd( "C:\Program Files" )]

### Description

The **cd** function changes FileMaker's working directory. This is the only means of changing the starting directory for commands executed by the **exec** function.

If the **directory** parameter is a partial path, it is interpreted relative to the working directory in effect prior to calling **cd**.

[ *Technical Note: The **cd** function does not perform expansion. See Appendix A. ]*

## yooShell\_getEnv( name )

### Parameters

**name** — The name of the environment variable to query.

### Data Returned to FileMaker

The current value of the given environment variable.

### Examples

```
Set Field [myDB::myText; yooShell_getEnv( "USER" )]
```

```
Set Field [myDB::myText; yooShell_getEnv( "USERNAME" )]
```

### Description

Use the **getEnv** function to inspect environment variables. The values returned by **getEnv** reflect the values that will be available to all commands executed using the **exec** function. These values can only be changed using the **setEnv** function.

The *environment* is a collection of named values. The behavior of many software packages is influenced by the particular values assigned to specific variables in the environment.

## yooShell\_setEnv( name; value )

### Parameters

**name** — The name of the environment variable to set.

**value** — The new value for the environment variable.

### Data Returned to FileMaker

No data is returned to FileMaker.

### Examples

```
Set Field [myDB::myVoid; yooShell_setEnv( "MYDATA", "foobar" )]
```

### Description

Use the **setEnv** function to create or modify environment variables. The values set by **setEnv** will be available to all commands executed using the **exec** function.

The *environment* is a collection of named values. The behavior of many software packages is influenced by the particular values assigned to specific variables in the environment.

Using **setEnv**, you can create your own environment variables to propagate information between calls to the **exec** function.

## yooShell\_config( option; [ value ] )

### Parameters

**option** — The predefined name of the setting to query or modify.

**value** — [optional] The value to assign to the setting. May also specify an action related to the setting.

### Data Returned to FileMaker

When querying a setting, the current value of the queried setting is returned. When modifying a setting, no data is returned.

### Examples

```
Set Field [myDB::myText; yooShell_config( "version"; )]
```

```
Set Field [myDB::myVoid; yooShell_config( "line-breaks"; "raw" )]
```

### Description

The **config** function controls various settings that impact the behavior of yooShell. Most users will normally not need to call the **config** function, as the default settings usually yield the desired behavior.

To query a setting, pass only the **option** parameter, and omit the **value** parameter. To modify a setting, pass the desired value in the **value** parameter.

Available <b>option</b> parameters	Accepted <b>value</b> parameters
"version" – Query this setting to get the version number of yooShell. This read-only setting cannot be modified.	n/a
"line-breaks" – Different computer software follows different conventions for how to represent the breaks between one line of text and the next. This setting controls whether yooShell translates line breaks to FileMaker's native format, or preserves the raw format of the text.	"translate" : <b>[default]</b> Line breaks will be translated to FileMaker's native format.
	"raw" : Line breaks will remain in their raw, untranslated format. This may cause text to be displayed improperly by FileMaker.

# Appendix A : Tips & Tricks

---

## Redirection

In addition to the *standard output* stream that is captured by the **exec** function, many commands will produce additional information in a separate stream known as *standard error*. Using "redirection" allows you to control the destination of the data in these streams.

A redirection is specified using the > operator. When specifying a redirection, the standard output stream is identified by the number 1, and standard error by the number 2.

To capture standard error in addition to standard output, redirect standard error into standard output:

```
yooShell_exec( "com 2>&1" )
```

To capture standard error instead of standard output, redirect standard error into standard output, and discard standard output:

```
yooShell_exec( "com 2>&1 1>/dev/null" ) [ Mac OS X ]
```

```
yooShell_exec( "com 2>&1 1>nul" ) [ Windows ]
```

A stream can be redirected into a file instead of capturing it (note that a redirection without the numeric stream identifier applies to standard output):

```
yooShell_exec( "com >output.txt" )
```

By default, redirecting to a file overwrites the file. To append to the file, use the >> operator:

```
yooShell_exec( "com >>output.txt" )
```

[ Note that discarding a stream is actually achieved by redirecting to a special file, '/dev/null' on Mac OS X, and 'nul' on Windows. ]

## Multiple Commands

The **command** parameter to the **exec** function can actually specify multiple commands in a single parameter. This can be especially useful when you need to execute a chain of commands whose correct execution depends upon changing the working directory or modifying the environment *during* execution.

Specify multiple commands in a single **command** parameter by separating them with a semicolon on Mac OS X, or an ampersand on Windows:

```
yooShell_exec( "com1 ; com2 ; com3" ) [ Mac OS X ]
```

```
yooShell_exec( "com1 & com2 & com3" ) [ Windows ]
```

## Conditional Execution

When executing multiple commands as described in the preceding section, the execution of the commands can be controlled using boolean logic by replacing the command separators with the boolean operators && and ||, and grouping commands in parentheses.

To execute a second command only if the first succeeds:

```
yooShell_exec( "com1 && com2" )
```

To execute a second command only if the first fails:

```
yooShell_exec( "com1 || com2" )
```

Commands can be grouped, and the logic compounded:

```
yooShell_exec( "com1 && ( com2 || com3 )" )
```

## Expansion

Special characters such as \* and ~ are expanded in a system-appropriate fashion by the **exec** function. Although expansion is not performed by the other functions in yooShell, you can use the **exec** function in conjunction with other functions to perform expansion when necessary.

Consider the example of changing the working directory to the user's home directory on Mac OS X:

```
yooShell_cd( yooShell_exec( "echo -n ~" ) )
```

---

All material is copyright © 2005 yooPlugs. All rights reserved.